(1) (a) Suppose predicates female(X), male(X), mother(X,Y), and father(X,Y) (the last means that X is the father of Y) have already been defined, in place of the “one-shot facts” in section 16.6 of the Sebesta text. (Namely, Section 16.6 has a list beginning female(shelley). male(bill).—ignore that, and note other differences between the following and the text’s example.) Use these predicates to define the following as Prolog predicates. Use additional variables Z,W,... as needed. Also say which relations are symmetric, and clarify the meaning of any relation that is not symmetric—like I did for father above.

Warning: The above is a different “axiom set” from the scheme of definitions used in the example file family.prolog, now in the directory /.../PROLOG/LECREC05/. (6+3+3+6+6 = 24 pts.) For 9 pts. extra credit, use negation to code halfSibling(X,Y) meaning that X and Y share one parent but are not full siblings.

(a) fullSibling(X,Y) (defined by sharing /both/ parents)
(b) nephew(X,Y)
(c) granddaughter(X,Y)
(d) firstCousin(X,Y)
(e) descendant(X,Y) (saying "... :- ancestor(Y,X)." is not allowed)

(2) Write the “ascenders” function in Prolog. Note that unlike ML, Prolog allows you to use a variable X twice in a pattern to match cases where the two occurrences are the same element—in case this helps you. Show the result of ascenders([1,4,4,5,1,8,8,8,3,9],Z), with the answer appearing in the accumulation parameter Z. (12 pts.)

(3) Consider the following basic Prolog predicates:

```prolog
class(C) ; C is a class
super(C1,C2) ; C1 is the immediate superclass of C2
hasMethod(C,M) ; C class has method M---in the strict sense of defining it
hasParams(C,M,L) ; L is the parameter list of method M in class C
listEqual(L1,L2) ; L1 equals list L2
hasReturn(C,M,D) ; D is the return type (class) of method M in class C
```

In terms of these predicates, write Prolog definitions of the following predicates:

```prolog
javaOverride(C,D,M) ; Method M in class D directly overrides M in C by Java
CsharpOverride(C,D,M) ; rules, respectively by C#'s overriding rules.
```

You may define helper predicates as needed, e.g. extending super to a predicate ancestor to cover non-immediate superclasses. Note that you must ensure that no other class between C and D defines method M. You may assume there is no overloading within classes—i.e., no cases like on Assignment 4 where class Derived had two versions of foo/Foo. You do not need to care (in C#) whether the virtual and/or override keywords have been specified. (24 pts., for 60 regular-credit points on this set)